Incremental Data Flow analysis using PRISM

Rashmi Rekha Mech
(*Project Guide: Prof. Uday Khedker*)

Department of Computer Science and Engineering,
Indian Institute of Technology, Bombay

June 2015

Outline of the talk

- Incremental Data Flow Analysis
    - Bit-vector Frameworks
    - General Frameworks
- Method to Reduce the Size of Affected Region
- Overview of PRISM
- Incremental Solver for PRISM
    - Architecture
    - Testing
- Conclusion

Part I

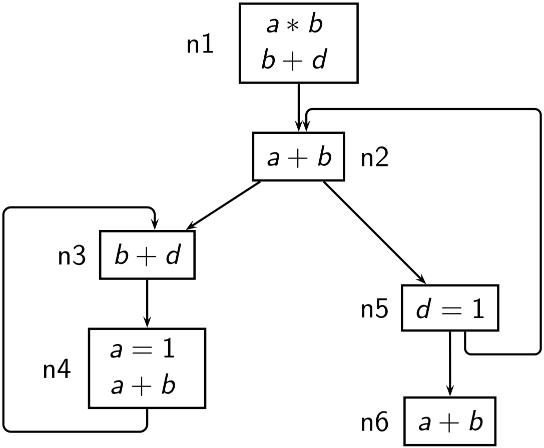
Incremental Data Flow Analysis

Why Incremental Analysis?

When program undergoes changes:

- Some or all computed data flow information becomes invalid
- Re-computation is required

Motivating Example - Available Expression Analysis

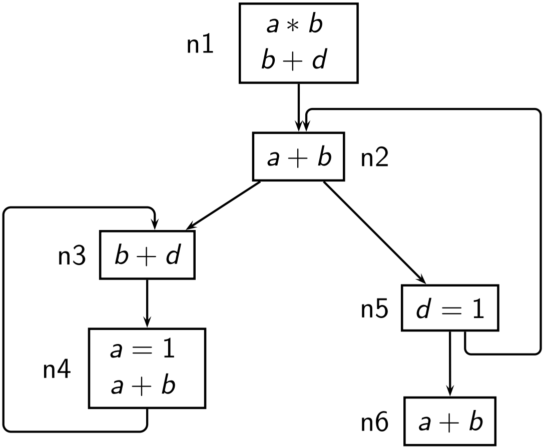


Bit Vector

$a * b$	$b + d$	$a + b$

Motivating Example - Available Expression Analysis

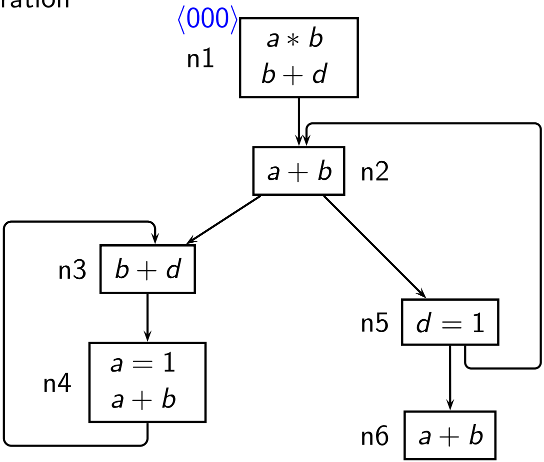
1st Iteration



Bit Vector

$a * b$	$b + d$	$a + b$

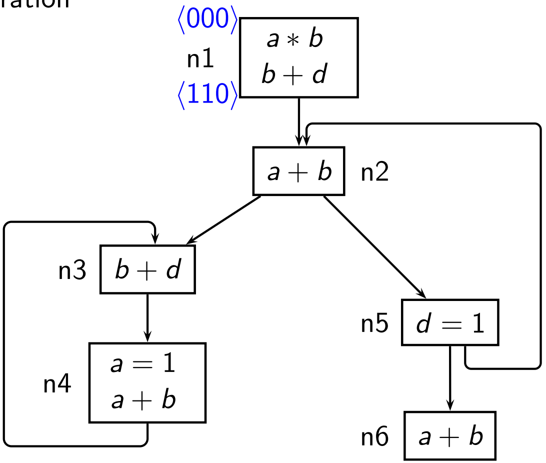
1st Iteration



Bit Vector	$a * b$	$b + d$	$a + b$

Motivating Example - Available Expression Analysis

1st Iteration

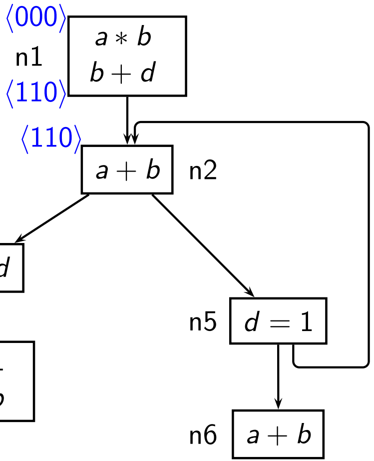


Bit Vector

$a * b$	$b + d$	$a + b$

Motivating Example - Available Expression Analysis

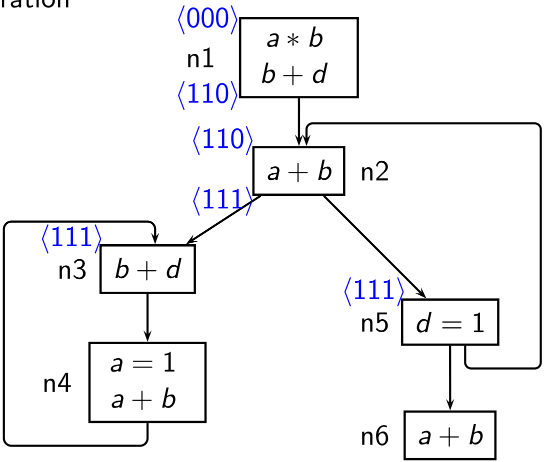
1st Iteration



Bit Vector $a * b$ | $b + d$ | $a + b$

Motivating Example - Available Expression Analysis

1st Iteration

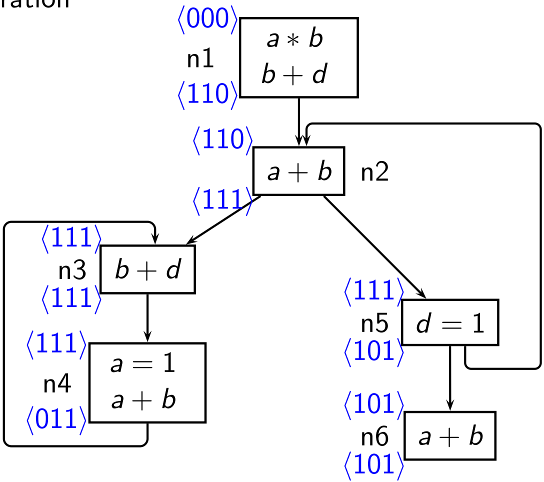


Bit Vector

$a * b$	$b + d$	$a + b$

Motivating Example - Available Expression Analysis

1st Iteration

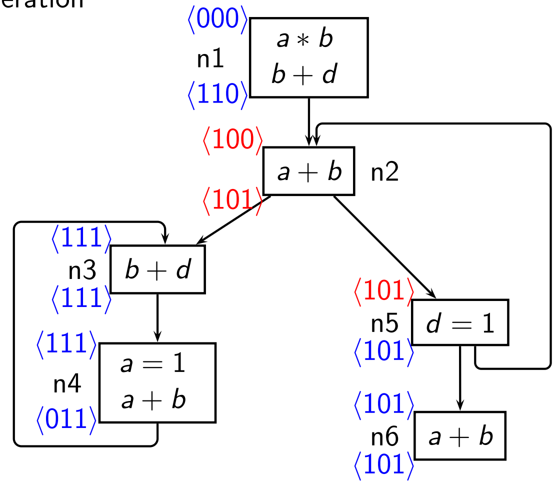


Bit Vector

$a * b$	$b + d$	$a + b$

Motivating Example - Available Expression Analysis

2nd Iteration

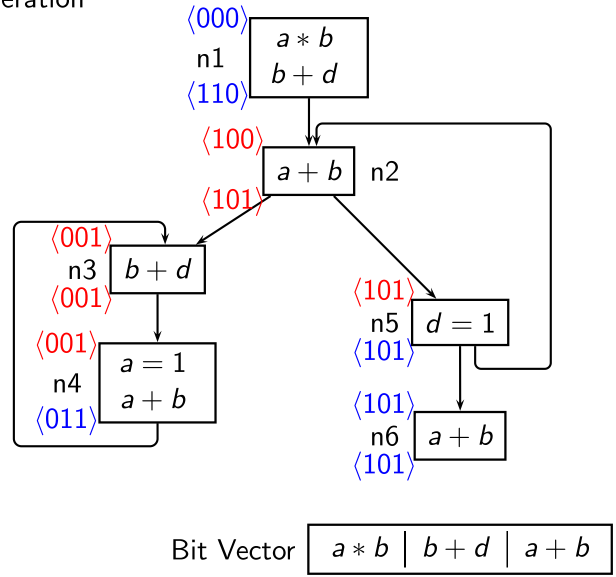


Bit Vector

$a * b$	$b + d$	$a + b$

Motivating Example - Available Expression Analysis

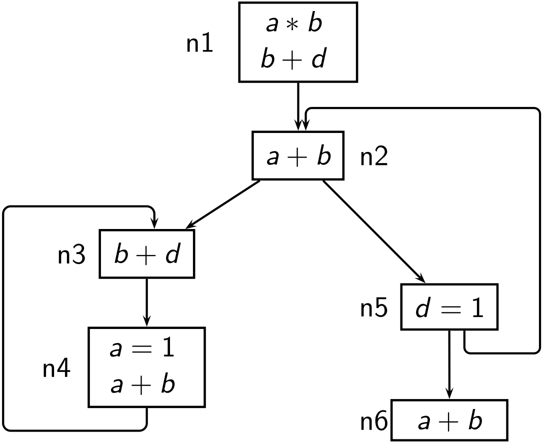
2nd Iteration



Motivating Example - Available Expression Analysis

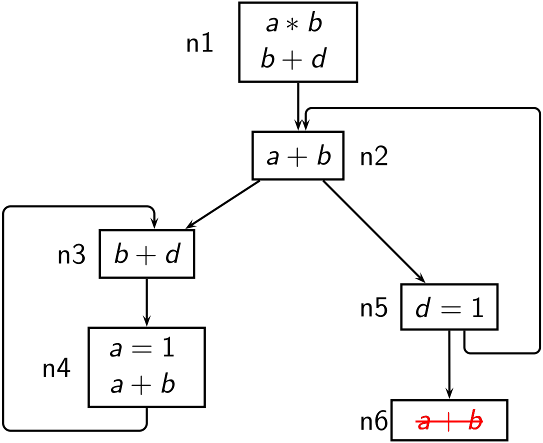
- It requires 3 iterations to converge

Motivating Example - Available Expression Analysis



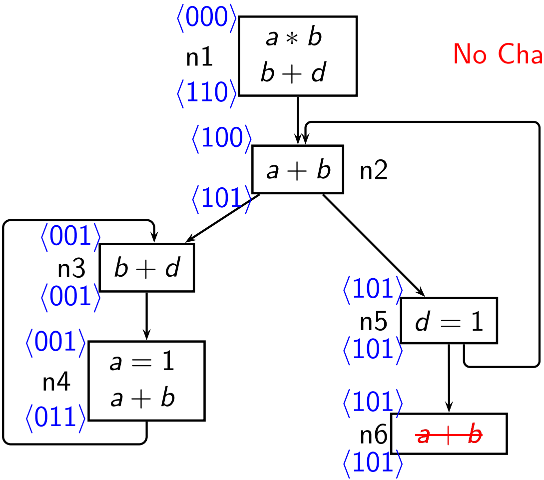
Bit Vector	$a * b$	$b + d$	$a + b$

Motivating Example - Available Expression Analysis



Bit Vector			
	$a * b$	$b + d$	$a + b$

Motivating Example - Available Expression Analysis



No Change in data flow values

Bit Vector	$a * b$	$b + d$	$a + b$

Motivating Example - Available Expression Analysis

- May unnecessarily analyze unaffected program behaviours which leads to redundant computation of old values which is very inefficient.
- Need an incremental analysis:
  - modifies only affected data flow information
  - more cost effective than **exhaustive** analysis

Part II

Incremental Analysis for Bit-vector Frameworks

Possible changes

- Due to program change, the following changes are possible[1]:
  - Change in flow functions
  - Change in control flow graph
  - Change in lattice

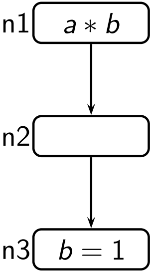
[1] Following slides talk about a change in flow function

Flow Functions in Bit-vector Frameworks

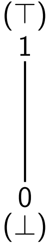
- Possible flow functions:
    - Raise : Result is always top
    - Lower : Result is always bottom
    - Propagate : Propagates the value from one program point to another

Example for Flow Functions

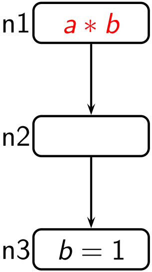
Available Expression Analysis



Lattice



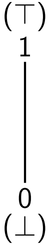
Available Expression Analysis



Raise Function

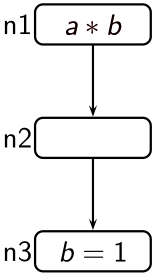
$\text{Gen}_1 = 1$
$\text{Kill}_1 = 0$
$\text{IN}_1 = 0$
$\text{OUT}_1 = \text{Gen}_1 \cup (\text{IN}_1\text{-Kill}_1) = 1$

Lattice



Example for Flow Functions

Available Expression Analysis

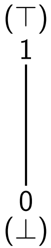


Raise Function

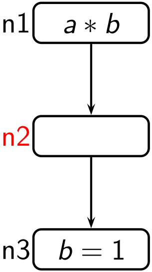
$\text{Gen}_1 = 1$
$\text{Kill}_1 = 0$
$\text{IN}_1 = 0$
$\text{OUT}_1 = \text{Gen}_1 \cup (\text{IN}_1 \text{-} \text{Kill}_1) = 1$

Result is always top

Lattice



Available Expression Analysis



Propagate Function

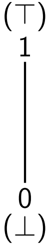
$\text{Gen}_2 = 0$

$\text{Kill}_2 = 0$

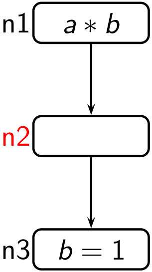
$\text{IN}_2 = 1$

$\text{OUT}_2 = \text{Gen}_2 \cup (\text{IN}_2\text{-Kill}_2) = \text{IN}_2 = 1$

Lattice



Available Expression Analysis

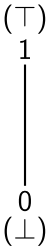


Propagate Function

$\text{Gen}_2 = 0$
$\text{Kill}_2 = 0$
$\text{IN}_2 = 1$
$\text{OUT}_2 = \text{Gen}_2 \cup (\text{IN}_2\text{-Kill}_2) = \text{IN}_2 = 1$

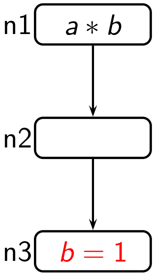
Propagates the value at IN to OUT

Lattice



Example for Flow Functions

Available Expression Analysis



Lower Function

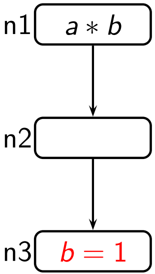
$\text{Gen}_3 = 0$
$\text{Kill}_3 = 1$
$\text{IN}_3 = 1$
$\text{OUT}_3 = \text{Gen}_3 \cup (\text{IN}_3\text{-Kill}_3) = 0$

Lattice

$(\top)$
1
0
$(\bot)$

Example for Flow Functions

Available Expression Analysis

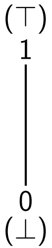


Lower Function

$\text{Gen}_3 = 0$
$\text{Kill}_3 = 1$
$\text{IN}_3 = 1$
$\text{OUT}_3 = \text{Gen}_3 \cup (\text{IN}_3\text{-Kill}_3) = 0$

Result is always bottom

Lattice

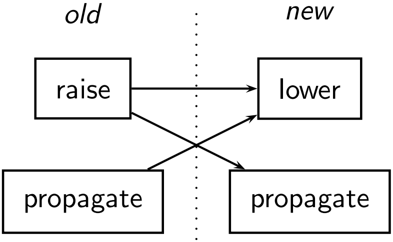


Possible Changes in Data Flow Values

- As a consequence of some change in a node, some data flow values may:
  - change from top to bottom
  - change from bottom to top
  - remain same

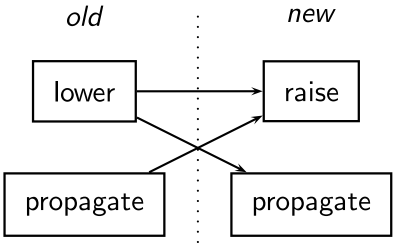
Top to Bottom Change

- Possible changes in flow functions.



Bottom to Top Change

- Possible changes in flow functions.

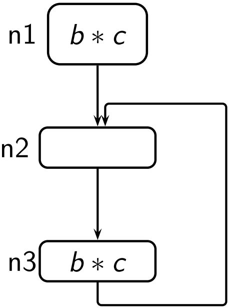


Handling Top to Bottom Change

- Bottom value is a final value even during analysis
- Whenever there is top to bottom change, the changes can be propagated directly to its neighbouring nodes

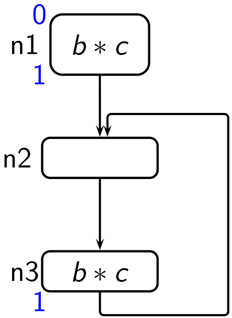
Example for Top to Bottom Change

Available Expression Analysis (Initial data flow values)



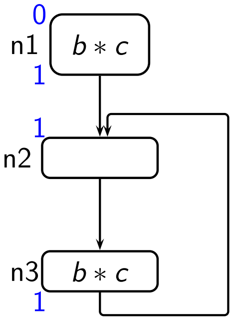
Example for Top to Bottom Change

Available Expression Analysis (Initial data flow values)

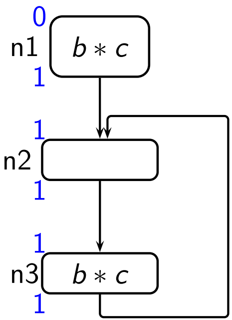


Example for Top to Bottom Change

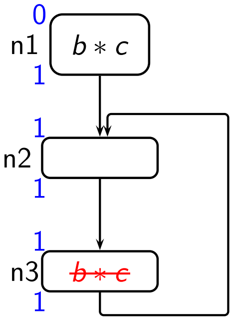
Available Expression Analysis (Initial data flow values)



Available Expression Analysis (Initial data flow values)

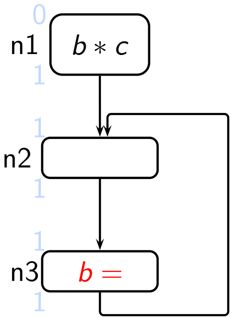


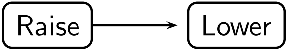
Example for Top to Bottom Change



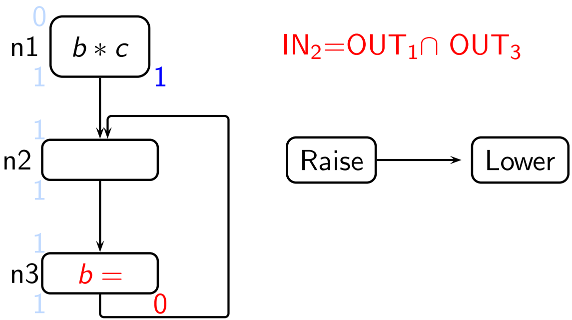
Example for Top to Bottom Change

Top to Bottom change





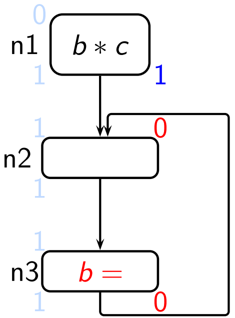
Top to Bottom change



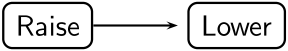
Directly Propagate the change to its neighbour

Example for Top to Bottom Change

Top to Bottom change



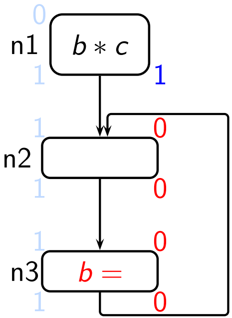
$IN_2 = OUT_1 \cap OUT_3$



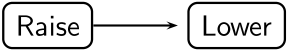
Directly Propagate the change to its neighbour

Example for Top to Bottom Change

Top to Bottom change



$IN_2 = OUT_1 \cap OUT_3$



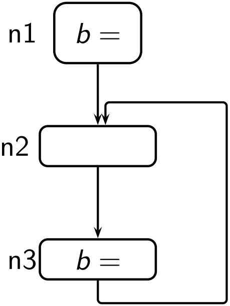
Directly Propagate the change to its neighbour

Handling Bottom to Top Change

- Top value is an intermediate value until data flow analysis is completed
- Whenever there is bottom to top change, we cannot directly propagate the changes to its neighbouring nodes
- Need some more processing

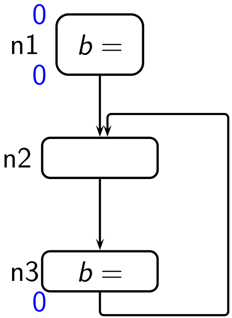
Example for Bottom to Top Change

Available Expression Analysis (Initial data flow values)



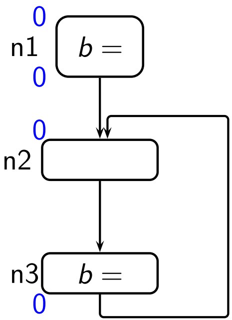
Example for Bottom to Top Change

Available Expression Analysis (Initial data flow values)



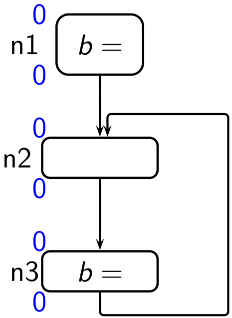
Example for Bottom to Top Change

Available Expression Analysis (Initial data flow values)

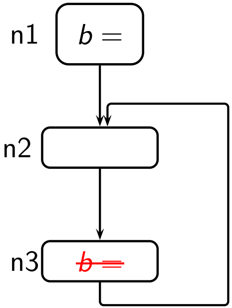


Example for Bottom to Top Change

Available Expression Analysis (Initial data flow values)

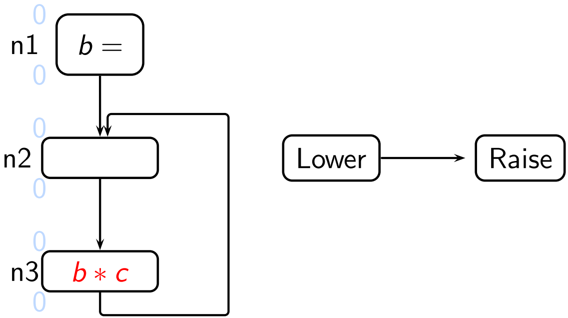


Example for Bottom to Top Change



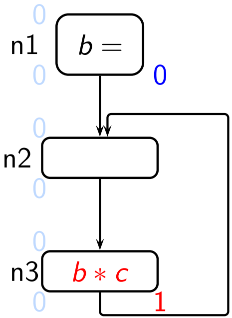
Example for Bottom to Top Change

Bottom to Top change

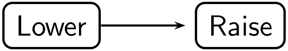


Example for Bottom to Top Change

Bottom to Top change



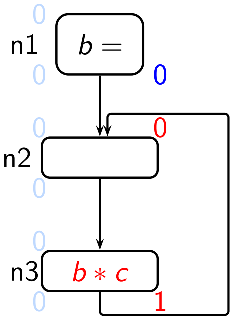
$IN_2 = OUT_1 \cap OUT_3$



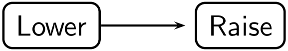
Cannot propagate the change to its neighbouring nodes

Example for Bottom to Top Change

Bottom to Top change



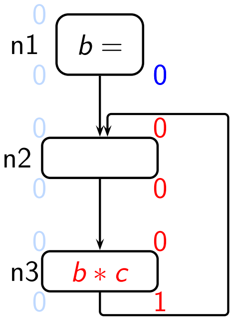
$IN_2 = OUT_1 \cap OUT_3$



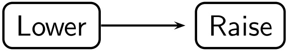
Cannot propagate the change to its neighbouring nodes

Example for Bottom to Top Change

Bottom to Top change



$IN_2 = OUT_1 \cap OUT_3$



Cannot propagate the change to its neighbouring nodes

Bottom to Top Change

- Need some more processing

Bottom to Top Change

- Steps to incorporate bottom to top change:

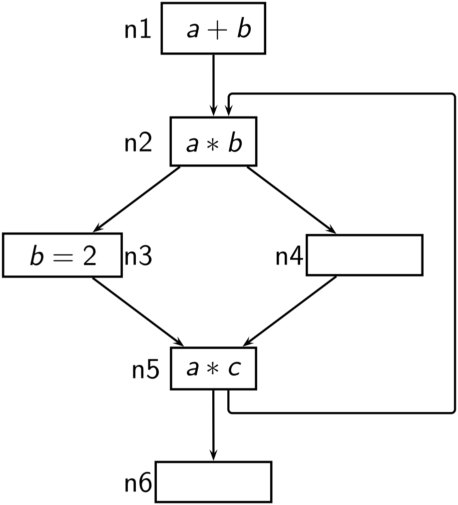
Bottom to Top Change

- Steps to incorporate bottom to top change:
  - Identify data flow values which may become top

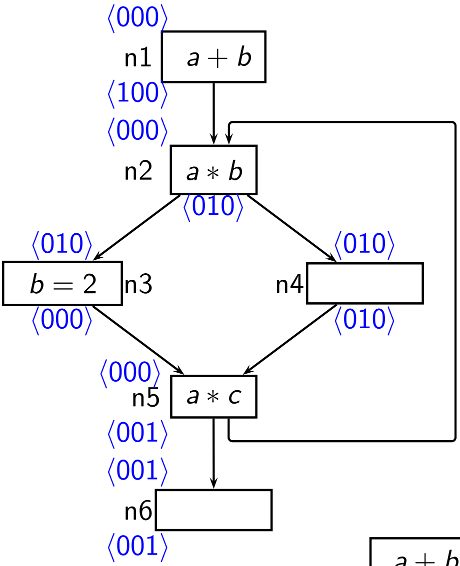
Bottom to Top Change

- Steps to incorporate bottom to top change:
  - Identify data flow values which may become top
  - Find out the data flow values which must remain bottom due to the effect of some other property

Motivating Example



$a + b$	$a * b$	$a * c$

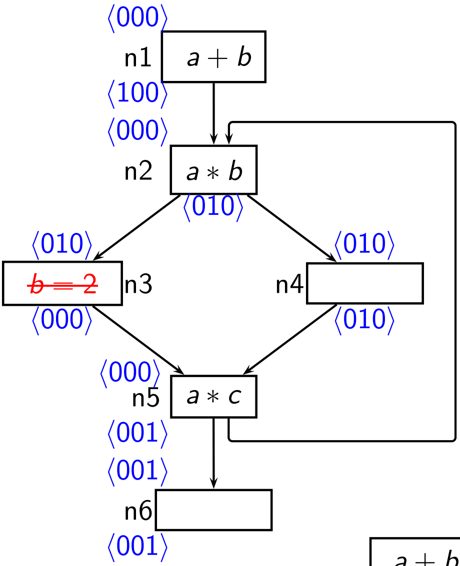


Initial Available Expression Analysis

	$a+b$		$a*b$		$a*c$	
Node	In	Out	In	Out	In	Out
1.	0	1	0	0	0	0
2.	0	0	0	1	0	0
3.	0	0	1	0	0	0
4.	0	0	1	1	0	0
5.	0	0	0	0	0	1
6.	0	0	0	0	1	1

$a+b$	$a*b$	$a*c$

Motivating Example

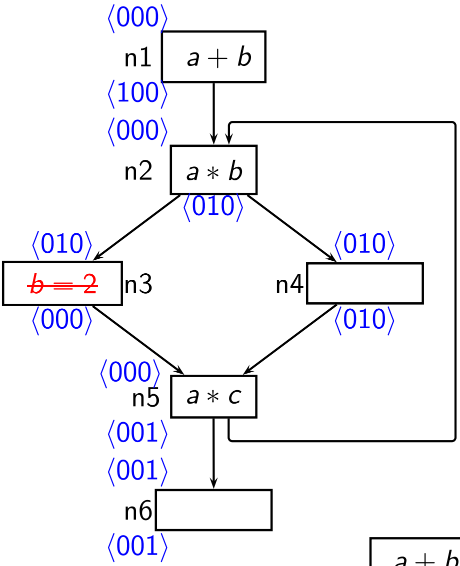


Initial Available Expression Analysis

	$a+b$		$a*b$		$a*c$	
Node	In	Out	In	Out	In	Out
1.	0	1	0	0	0	0
2.	0	0	0	1	0	0
3.	0	0	1	0	0	0
4.	0	0	1	1	0	0
5.	0	0	0	0	0	1
6.	0	0	0	0	1	1

$a+b$	$a*b$	$a*c$

Motivating Example



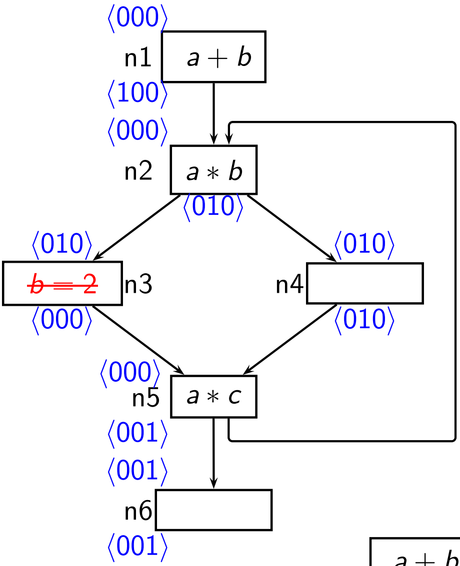
Initial Available Expression Analysis

lower

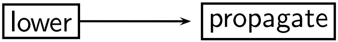
propagate

$a + b$	$a * b$	$a * c$

Motivating Example

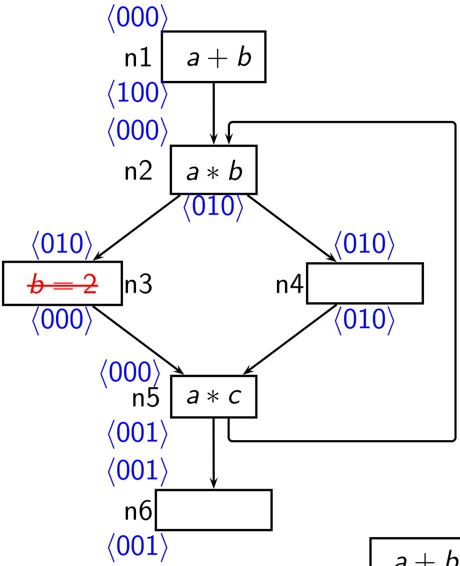


Initial Available Expression Analysis



$a + b$	$a * b$	$a * c$

Motivating Example



Initial Available Expression Analysis

$\boxed{\text{lower}} \longrightarrow \boxed{\text{propagate}}$

Bottom to top change

$a+b$	$a*b$	$a*c$

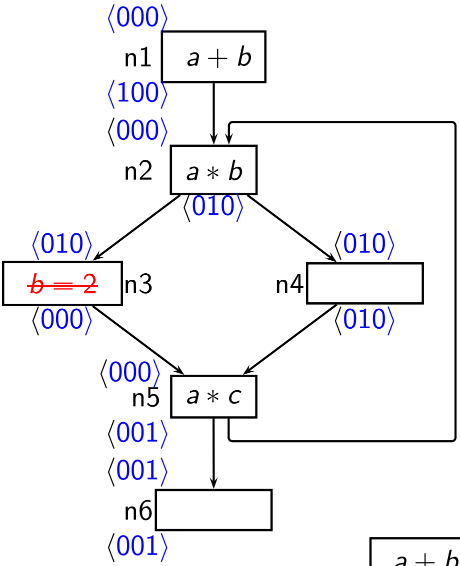
Motivating Example - Step 1

- To identify data flow values which were 0 and *may* become 1 due to this change

Motivating Example - Step 1

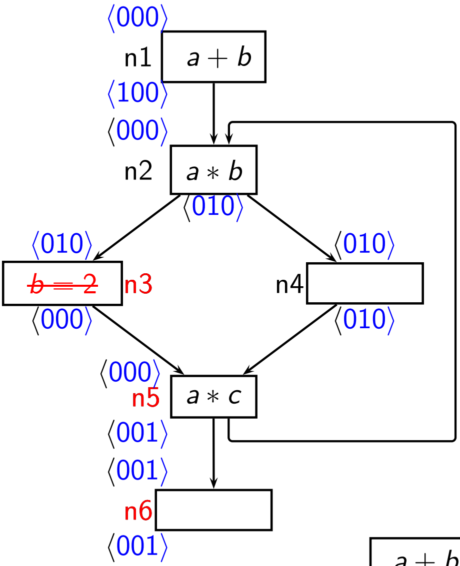
- To identify data flow values which were 0 and *may* become 1 due to this change
    - Affected region

Motivating Example - Step 1



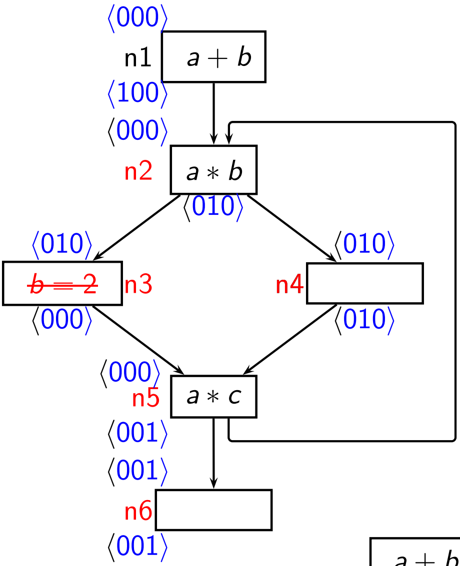
a + b	a ∗ b	a ∗ c

Motivating Example - Step 1



$a+b$	$a*b$	$a*c$

Motivating Example - Step 1

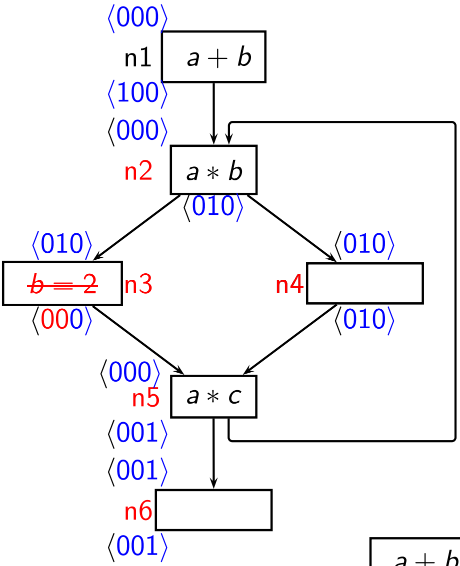


Affected Region

$\langle\ OUT_3,\ IN_5,\ OUT_5,\ IN_6,\ OUT_6,\ IN_2,\ OUT_2,\ IN_4,\ OUT_4,\ IN_3 \rangle$

$a+b$	$a*b$	$a*c$

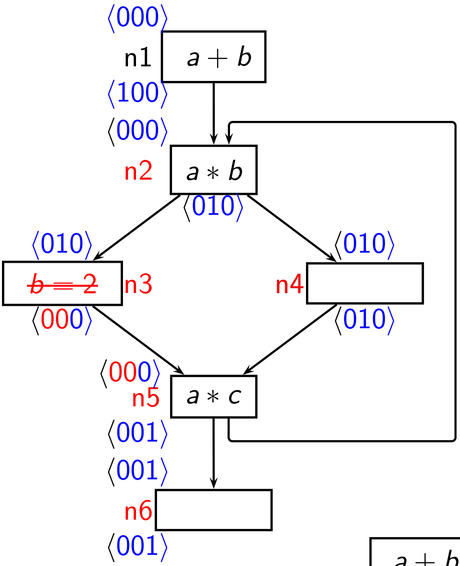
Motivating Example - Step 1



Data flow values which may become 1

$a+b$	$a*b$	$a*c$

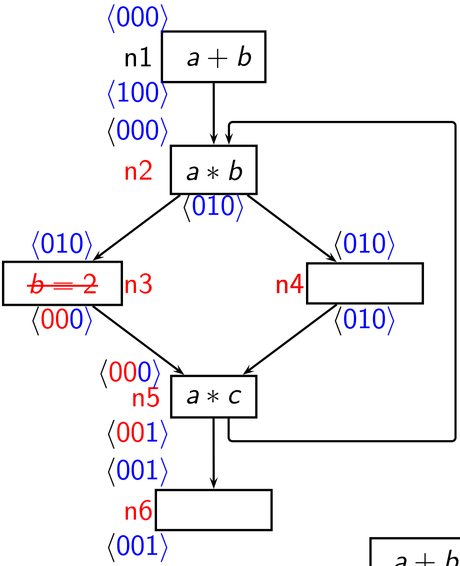
Motivating Example - Step 1



Data flow values which may become 1

$a + b$	$a * b$	$a * c$

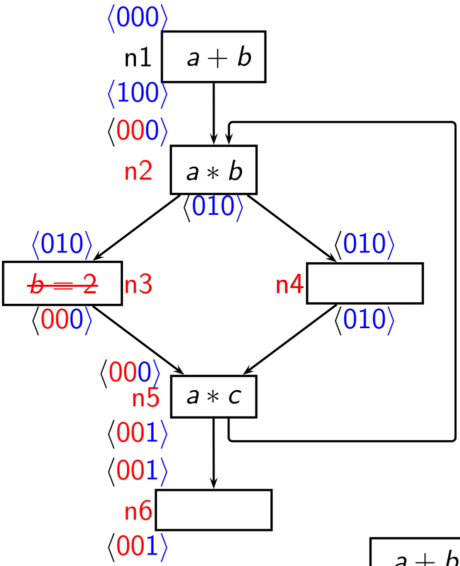
Motivating Example - Step 1



Data flow values which may become 1

a + b	a ∗ b	a ∗ c

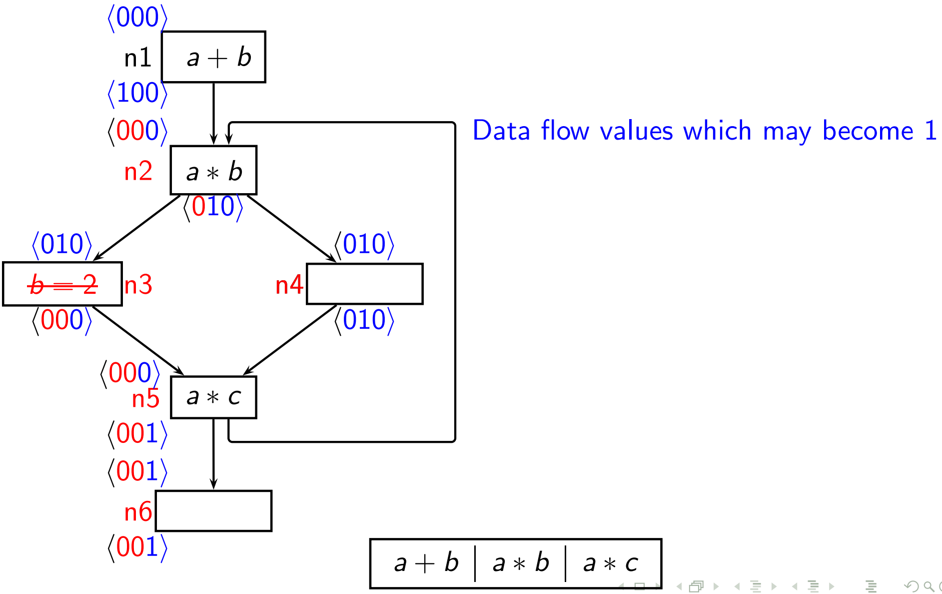
Motivating Example - Step 1



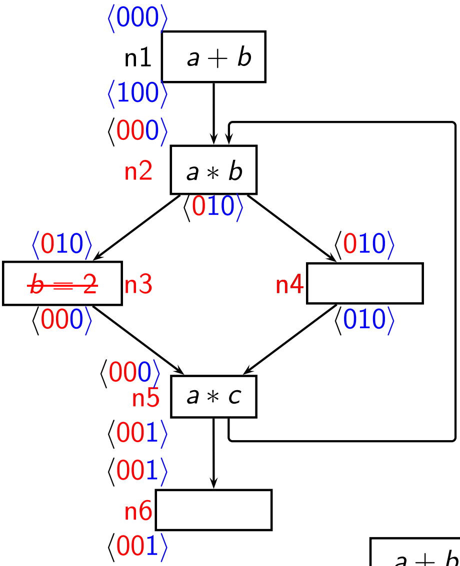
Data flow values which may become 1

$a+b$	$a*b$	$a*c$

Motivating Example - Step 1



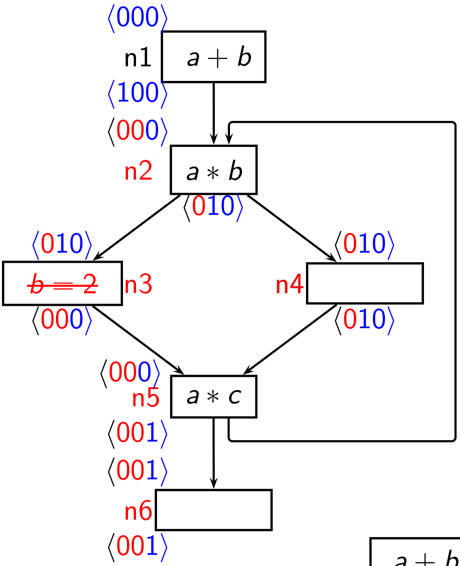
Motivating Example - Step 1



Data flow values which may become 1

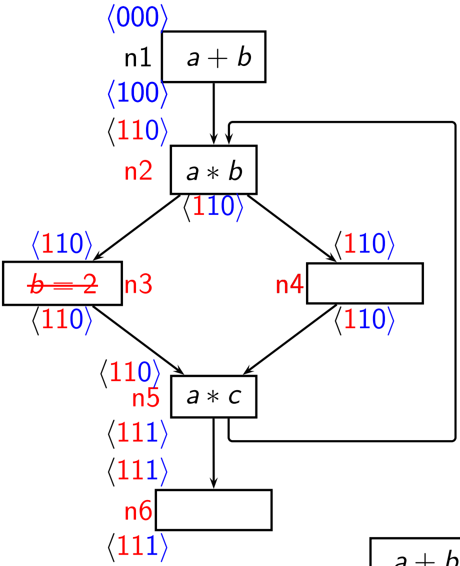
$a + b$	$a * b$	$a * c$

Motivating Example - Step 1



Data flow values which may become 1

$a+b$	$a*b$	$a*c$



Data flow values which may become 1

	$a + b$		$a * b$		$a * c$	
Node	In	Out	In	Out	In	Out
1.						
2.	1	1	1			
3.	1	1		1		
4.	1	1				
5.	1	1	1	1		
6.	1	1	1	1		

$a + b$	$a * b$	$a * c$

Motivating Example - Step 2

- To identify data flow values which must remain bottom due to the effect of some other properties

Motivating Example - Step 2

- To identify data flow values which must remain bottom due to the effect of some other properties
  - Initialize affected region to top.

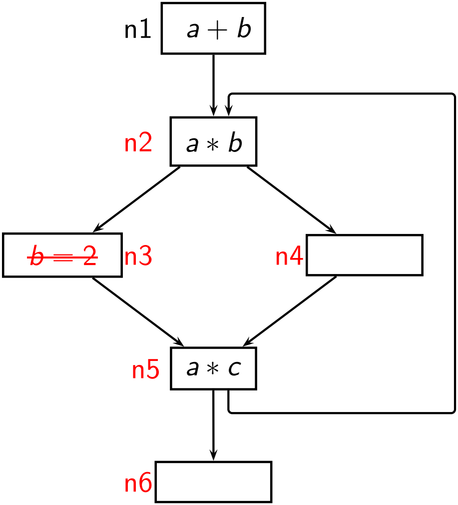
Motivating Example - Step 2

- To identify data flow values which must remain bottom due to the effect of some other properties
  - Initialize affected region to top.
  - Identify boundary nodes.

Motivating Example - Step 2

- To identify data flow values which must remain bottom due to the effect of some other properties
  - Initialize affected region to top.
  - Identify boundary nodes.
  - Compute values at boundary nodes and propagate them.

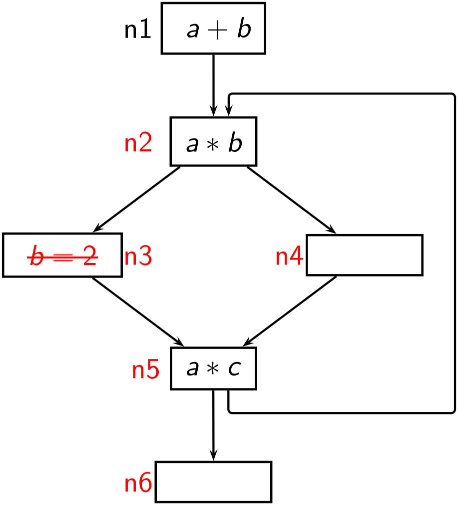
Motivating Example - Step 2



$a + b$	$a * b$	$a * c$

Motivating Example - Step 2

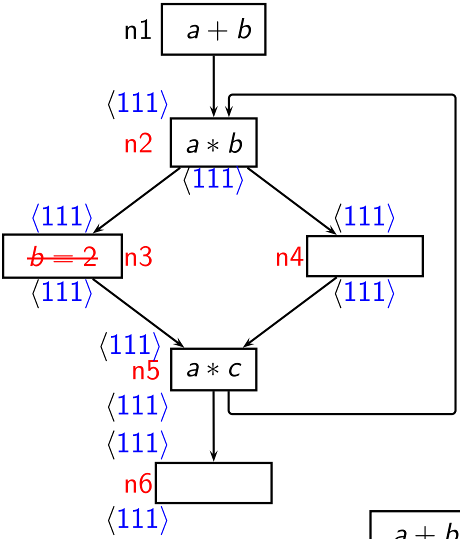
Initialize affected region to top



$a + b$	$a * b$	$a * c$

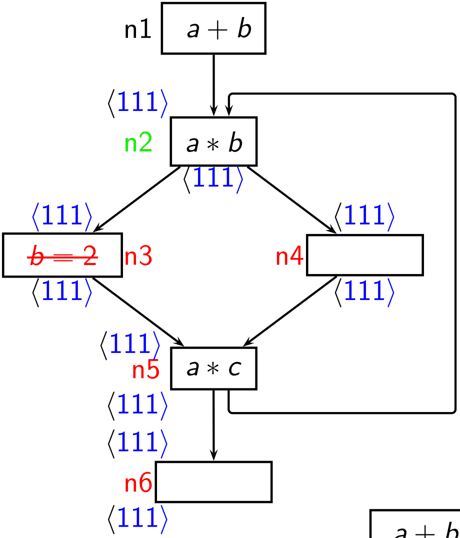
Motivating Example - Step 2

Initialize affected region to top



$a + b$	$a * b$	$a * c$

Motivating Example - Step 2

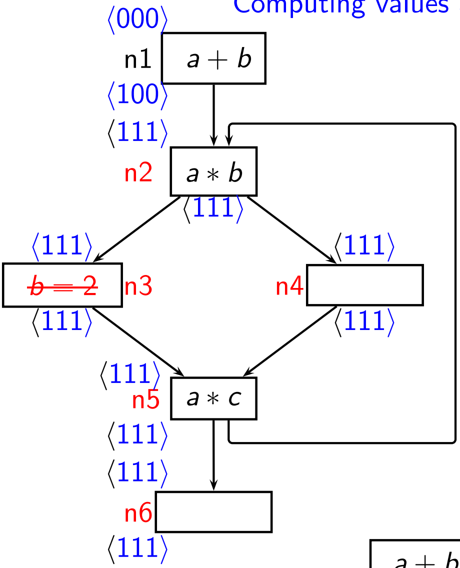


Node 2 is Boundary node

$a + b$	$a * b$	$a * c$

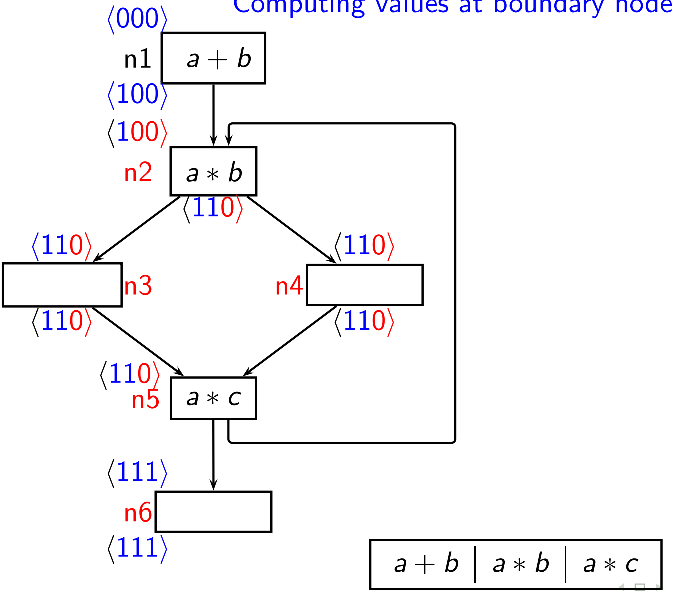
Motivating Example - Step 2

Computing values at boundary node and propagating them

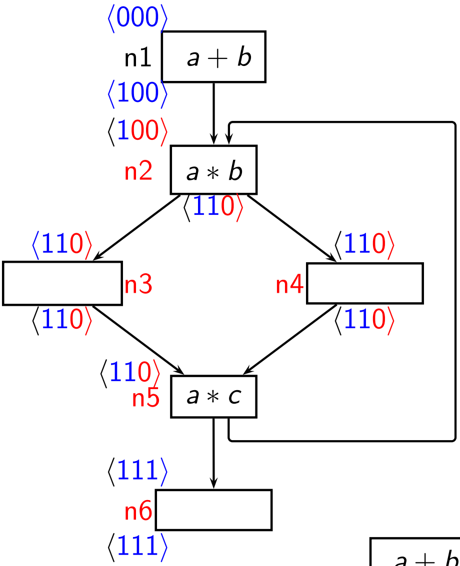


$a+b$	$a*b$	$a*c$

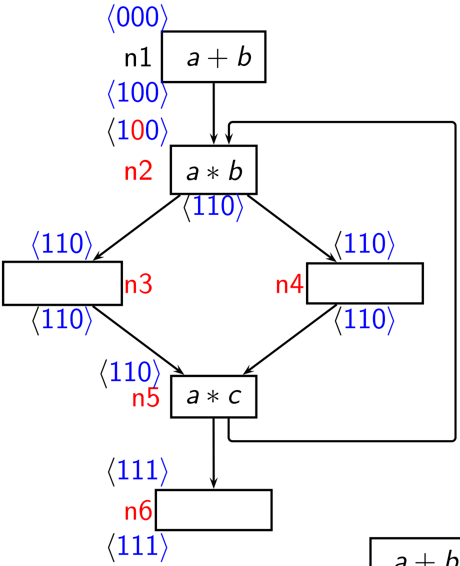
Computing values at boundary node and propagating them



Motivating Example - Step 2



a + b	a ∗ b	a ∗ c

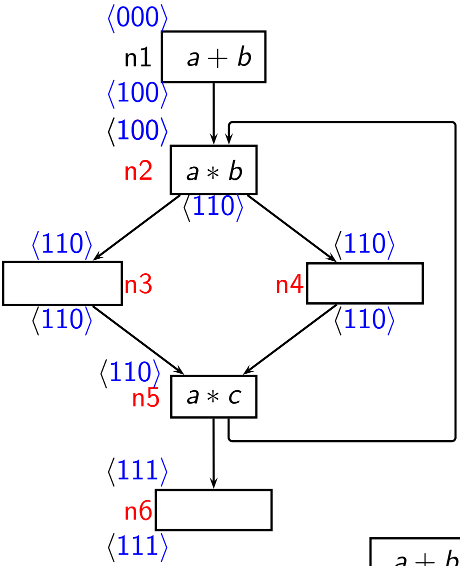


Values which must remain 0

	$a + b$		$a * b$		$a * c$	
Node	In	Out	In	Out	In	Out
1.						
2.			0			
3.						
4.						
5.						
6.						

$a + b$	$a * b$	$a * c$

Motivating Example - Step 2



Final values

	$a+b$		$a*b$		$a*c$	
Node	In	Out	In	Out	In	Out
1.	0	1	0	0	0	0
2.	1	1	0	1	0	0
3.	1	1	1	1	0	0
4.	1	1	1	1	0	0
5.	1	1	1	1	0	1
6.	1	1	1	1	1	1

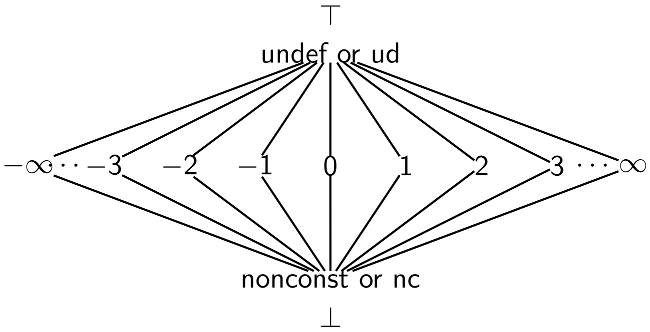
$a+b$	$a*b$	$a*c$

Part III

Incremental Analysis for General Frameworks

Incremental Analysis for General Frameworks

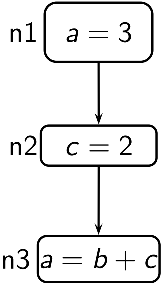
- Consider constant propagation analysis
- Component Lattice for Constant Propagation:



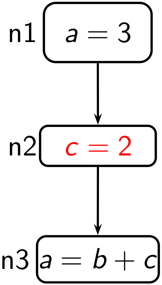
Flow Functions

- Possible flow functions
  - Top : Similar to raise function
  - Bottom : Similar to lower function
  - Constant : Always produce a constant value
  - Side level : Result depends on the operands of the expression

Constant Functions

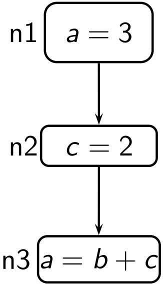


Constant Functions

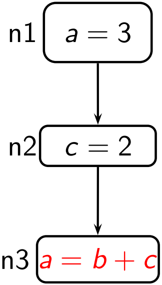


Produce constant value

Side Level Functions



Side Level Functions

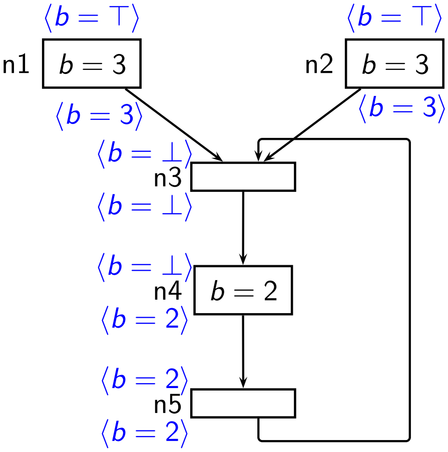


Result depends on the operands

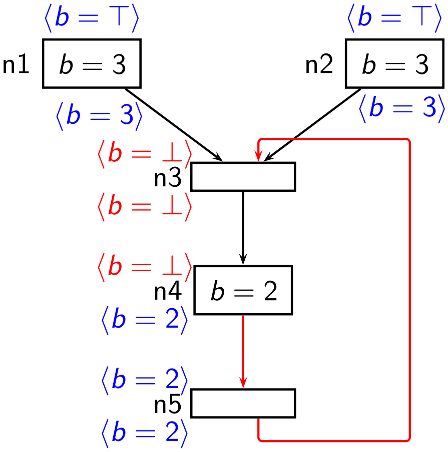
Issues in General Frameworks

- Unlike bit-vector frameworks, when there is a change to bottom:
  - we cannot propagate the change to its neighbouring nodes

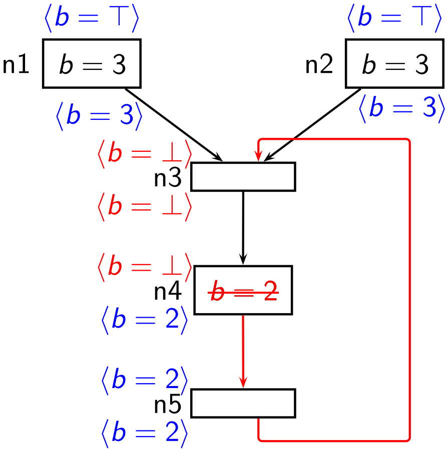
Issues in General Frameworks



Issues in General Frameworks

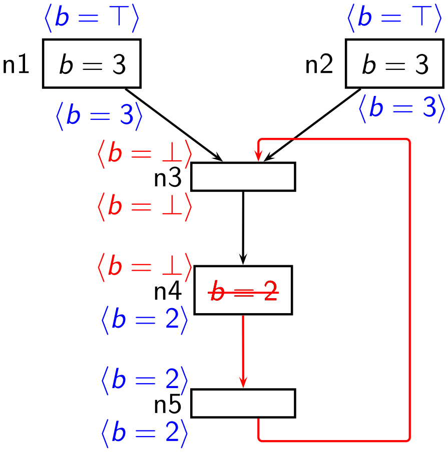


Issues in General Frameworks



Change to bottom

Issues in General Frameworks



We cannot propagate the change

Issues in General Frameworks

- Unlike bit-vector frameworks, we may need to create an affected region even if there is a change to bottom.
- Solution is to create affected region for all kind of changes.

Part IV

Method to Reduce the Size of Affected Region

Method to Reduce the Size of Affected Region

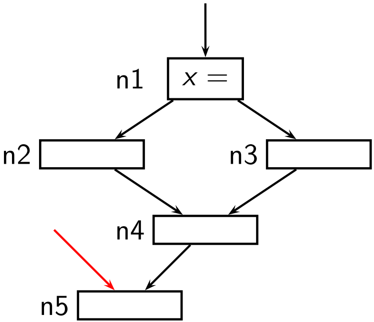
- Based on the observation that some boundary nodes can be characterized by the concept of **Dominance Frontier**.
- Eliminate some boundary nodes from being included in the affected region.

Method to Reduce the Size of Affected Region

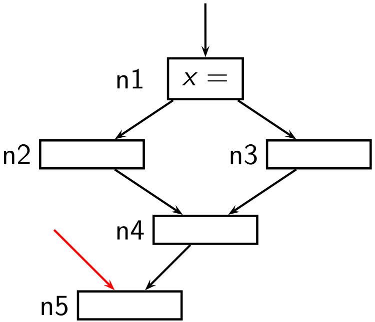
- Let n and m be nodes in CFG. The node n is said to *dominate* m ($n \geq m$), if every path from **Start** to m passes through n.
- If $n \neq m$, then n strictly dominates m, denoted as $n > m$
- **Dominance Frontier**:

$$df(n) = \{m \mid \exists p \in pred(m), (n \geq p \text{ and } n \not> m)\} \tag{1}$$

Method to Reduce the Size of Affected Region

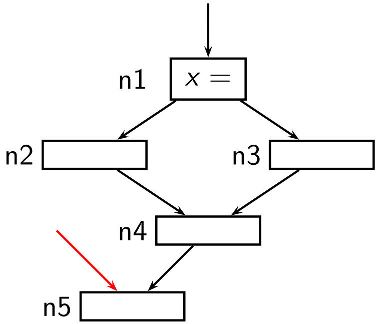


Method to Reduce the Size of Affected Region



$n1$ dominates $n4$

Method to Reduce the Size of Affected Region



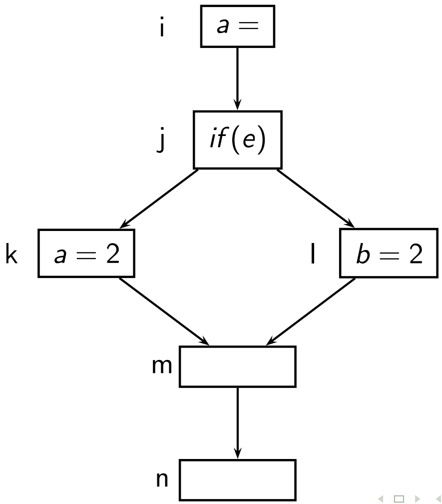
$n5$ is a dominance frontier of $n1$

Method to Reduce the Size of Affected Region

- All Dominance frontier nodes are boundary nodes (but not vice versa).

Method to Reduce the Size of Affected Region

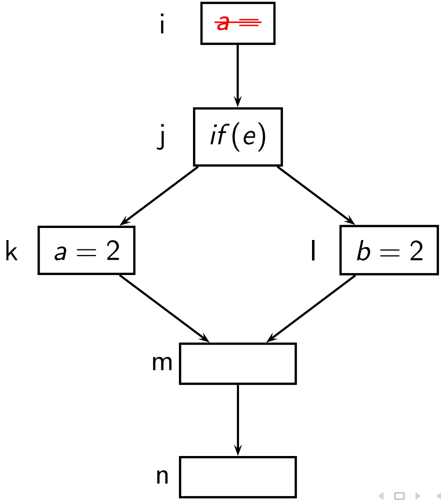
- All Dominance frontier nodes are boundary nodes (but not vice versa).



Method to Reduce the Size of Affected Region

- All Dominance frontier nodes are boundary nodes (but not vice versa).

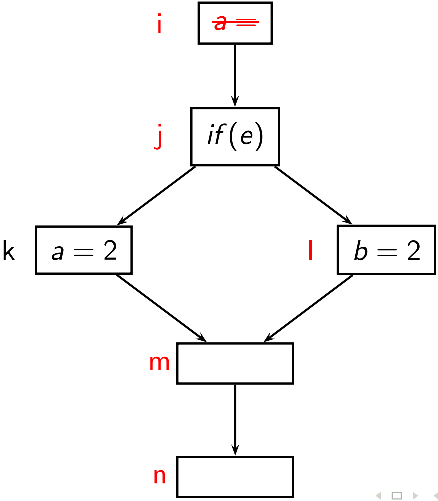
Affected region: $< i, j, l, m, n >$



Method to Reduce the Size of Affected Region

- All Dominance frontier nodes are boundary nodes (but not vice versa).

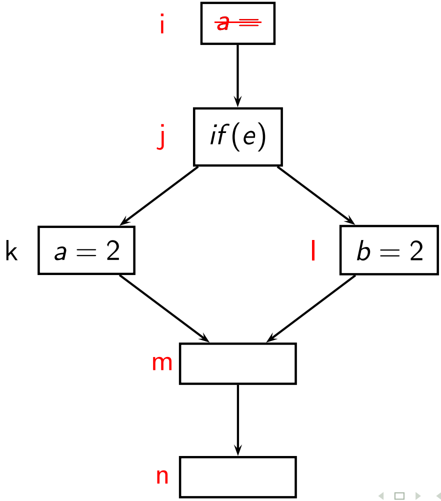
Affected region: $< i, j, l, m, n >$



Method to Reduce the Size of Affected Region

- All Dominance frontier nodes are boundary nodes (but not vice versa).

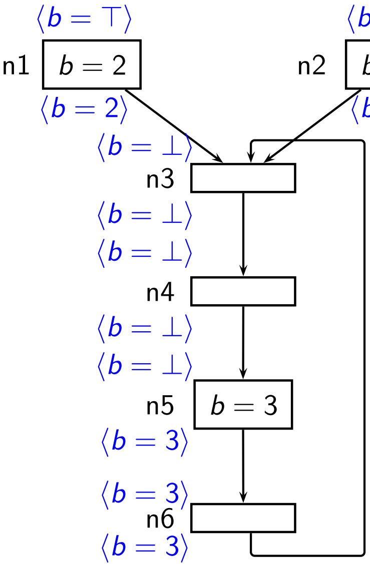
m is a boundary node and is dominated by i



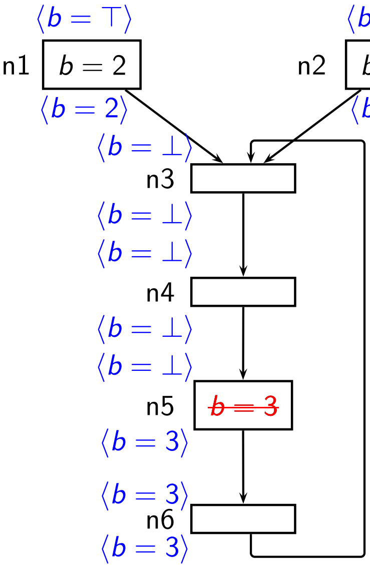
Method to Reduce the Size of Affected Region

- Possible removal candidates must be dominance frontier of changed node.

Method to Reduce the Size of Affected Region

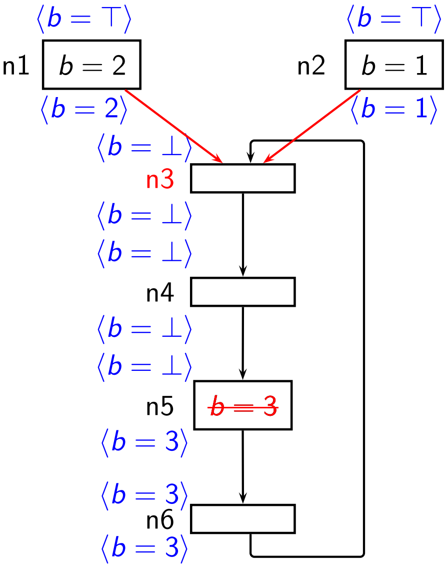


Method to Reduce the Size of Affected Region

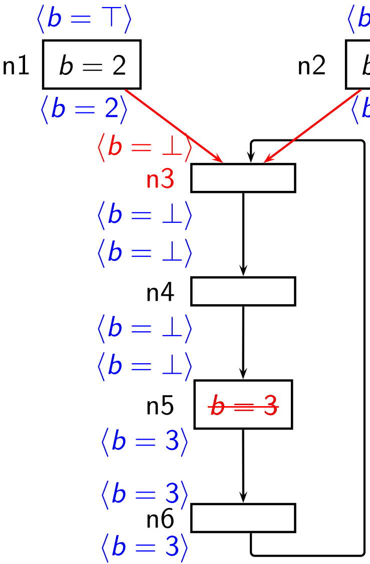


AR: $< n_5, n_6, n_3, n_4 >$

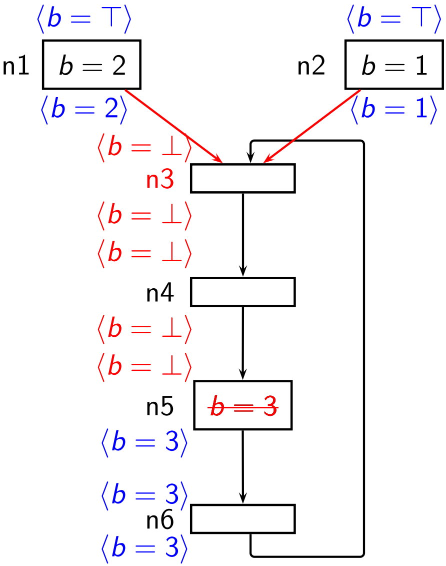
Method to Reduce the Size of Affected Region



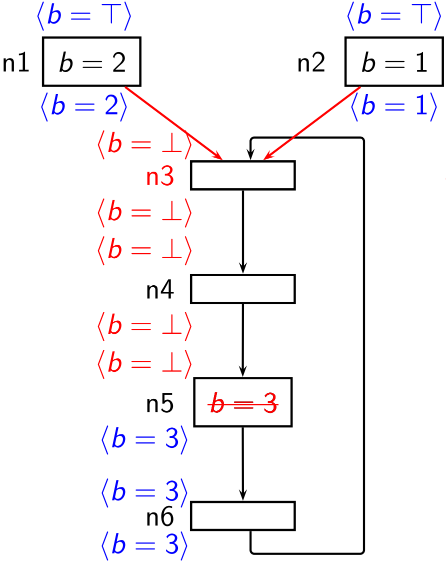
Method to Reduce the Size of Affected Region



Method to Reduce the Size of Affected Region



Method to Reduce the Size of Affected Region



can eliminate $n3$

Method to Reduce the Size of Affected Region

- Also applicable for bit-vector frameworks.

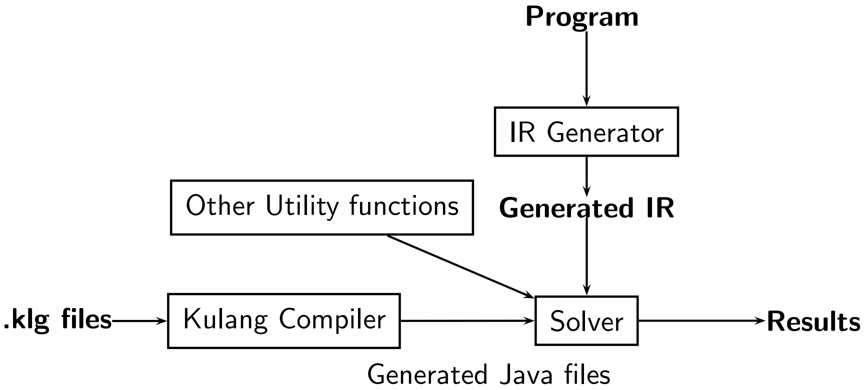
Part V

Overview of PRISM

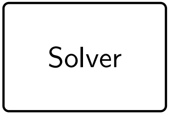
PRISM

- PRISM is a program analyzer generator developed by TATA Research Development and Design Center (TRDDC)

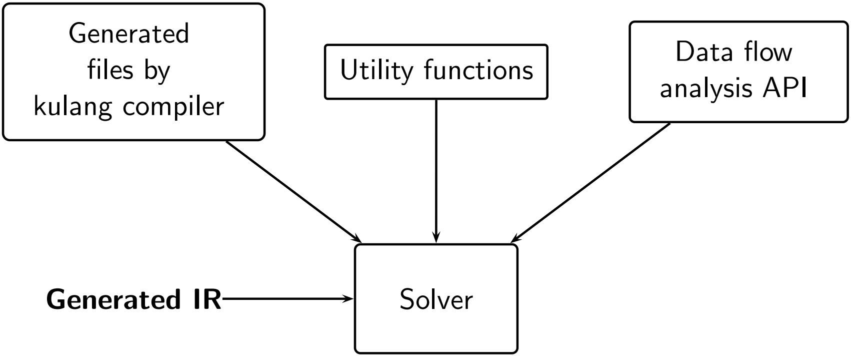
Old Architecture of PRISM



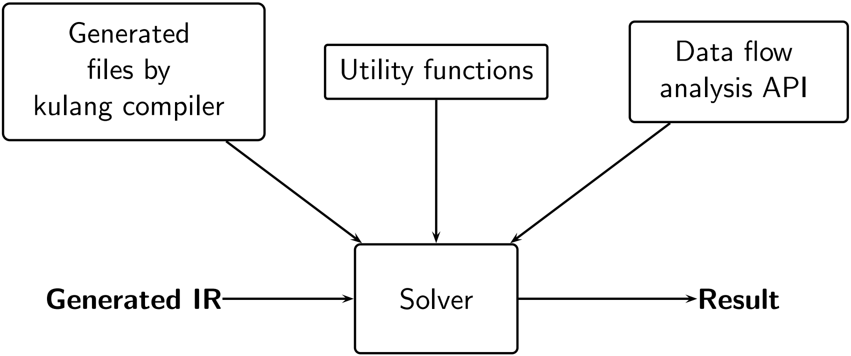
Architecture of Analyzer Generator



Architecture of Analyzer Generator



Architecture of Analyzer Generator



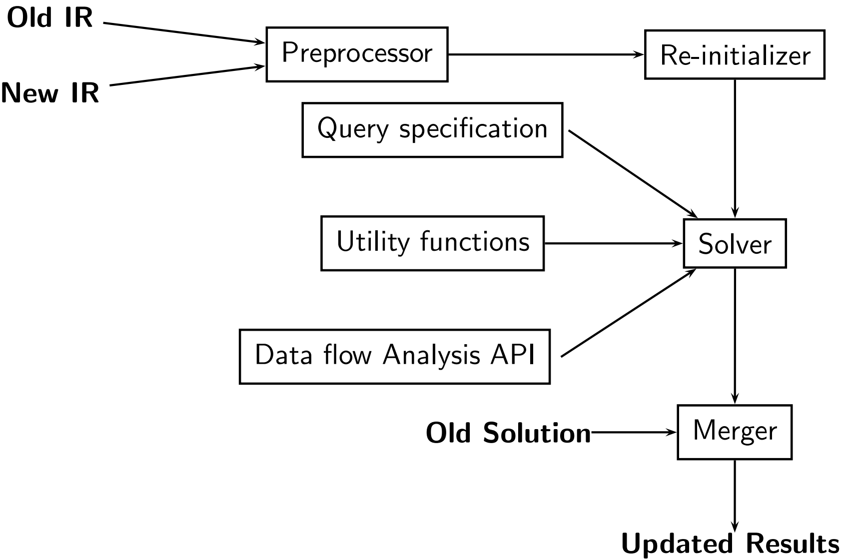
Part VI

Incremental Solver for PRISM

Architecture of Incremental Solver for PRISM

- Architecture of incremental solver for PRISM is divided into following parts:
  - Preprocessing: reads two IRs and find the difference between them.
  - Data Flow Computation:
    - Re-initialization: constructs affected region.
    - Re-computation: identify boundary nodes and compute the information.
  - Update: merge the information.

Architecture of Incremental Solver for PRISM



Assumptions

- Pointer information will remain same.
- No change in the context information.
- Declaration of variable hasn't change.
- The result of old solver is stored in non-standard format. So the following assumptions are made:
  - No structural change in the graph.
  - A name can refer to a single variable in a program at any given program point.
  - Past information is stored flow sensitively.

Limitations

- Following are current limitations:
  - Code for affected region has to be written manually.
  - Result is stored in a non-standard format by the Solver.
  - Non inclusion of global and local declarations.
- These can be overcome.

Part VII

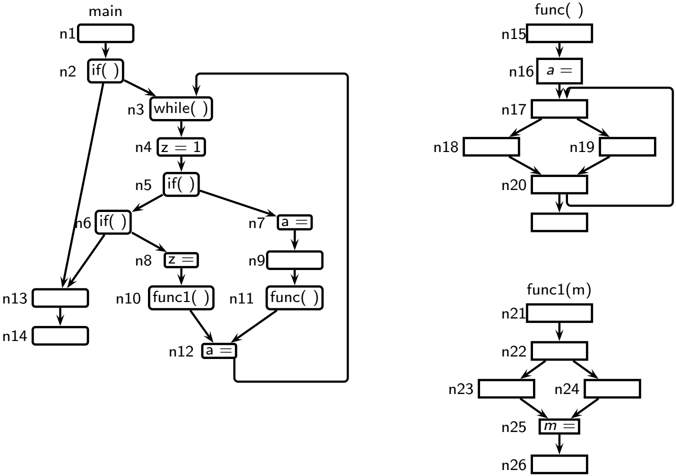
Testing

Testing

- In the absence of serialization of IR, it was not possible to test the code in real world applications.
- Artificially added the changes to check the performance of the incremental solver.

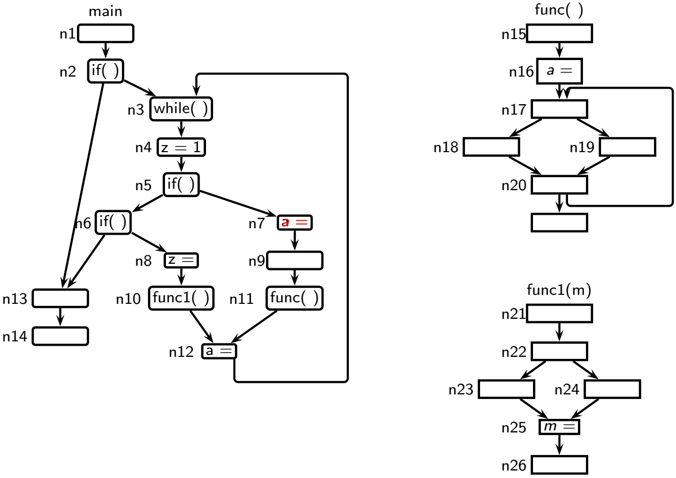
Test Case 1 : *a* is a global variable

n7 is a changed node



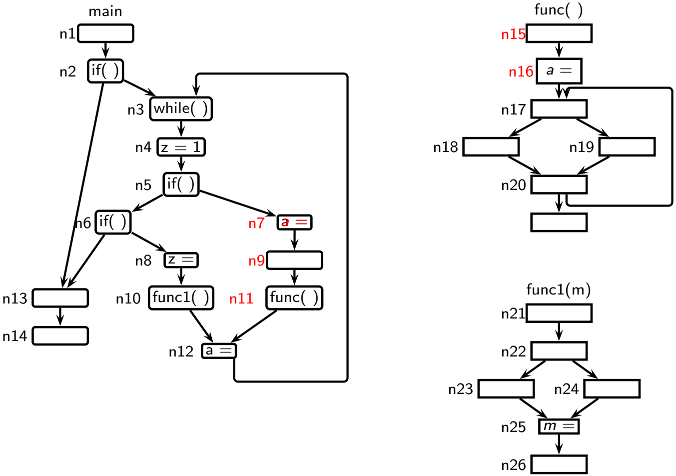
Test Case 1 : *a* is a global variable

n7 is a changed node



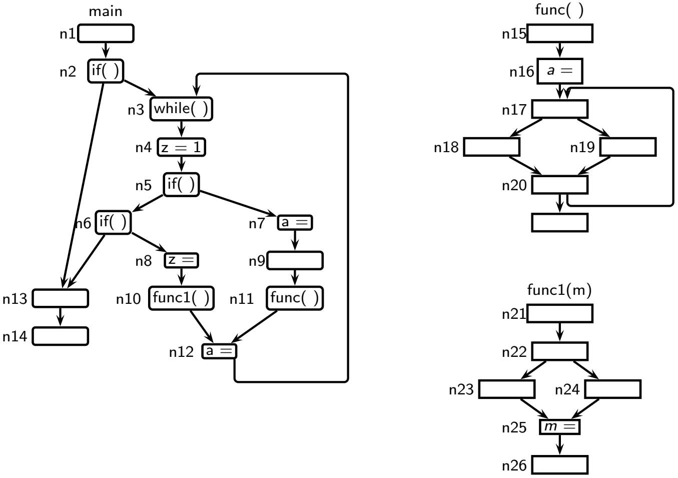
Test Case 1 : *a* is a global variable

*n*7 is a changed node

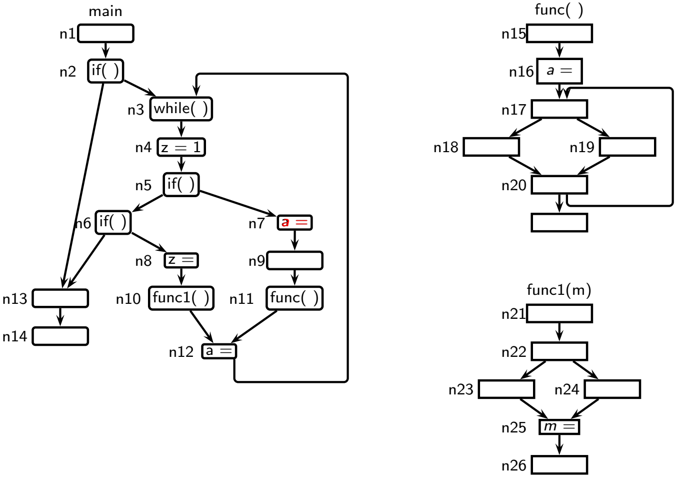


Test Case 2 : *a* is a local variable

n7 is a changed node

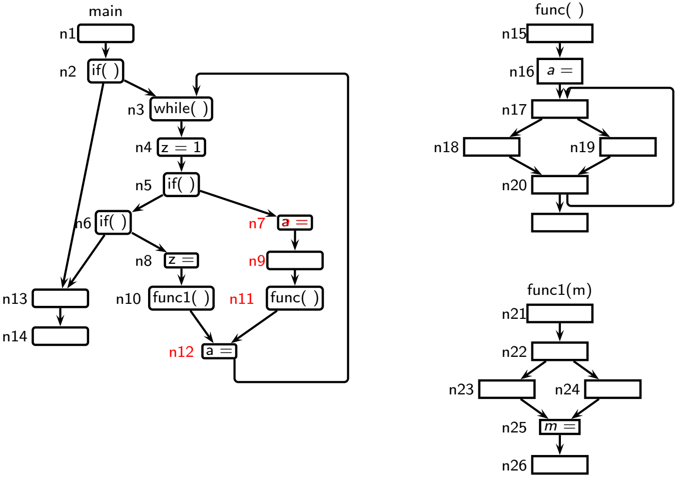


n7 is a changed node



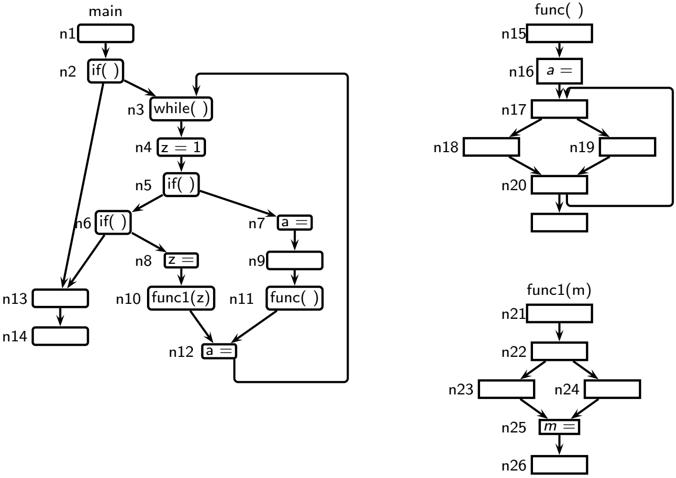
Test Case 2 : *a* is a local variable

n7 is a changed node

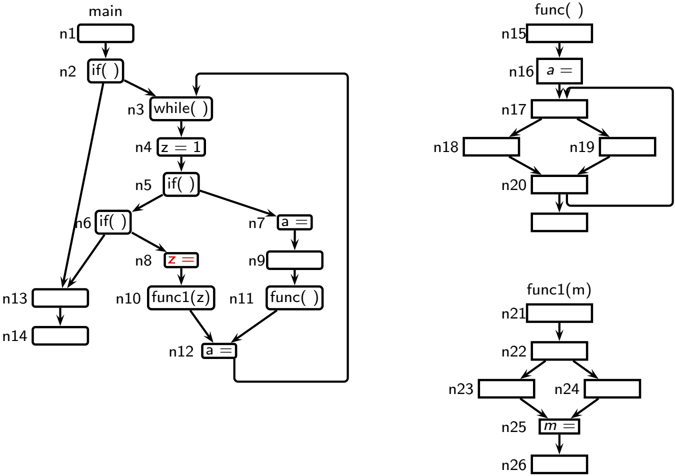


Test Case 3 : passed as a parameter

n8 is a changed node

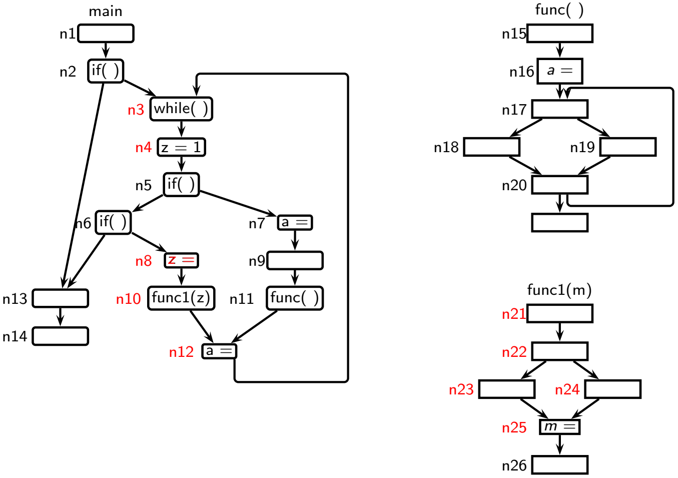


n8 is a changed node



Test Case 3 : passed as a parameter

n8 is a changed node



Part VIII

Future Work

Future Work

- Change the Kulang compiler to generate the code to compute affected region from the flow function.
- Extending the method of reducing the size of affected region for multiple changes.
- Persist the result of the solver.

Part IX

Thank You!